

Fundamentos de Algoritmos e Estrutura de Dados - Aula 07 - Complexidade e Programação Dinâmica

Prof. André Gustavo Hochuli

gustavo.hochuli@pucpr.br

aghochuli@ppgia.pucpr.br

Plano de Aula

- **Complexidade Computacional**
- **Notação BigO**
- **Programação Dinâmica**

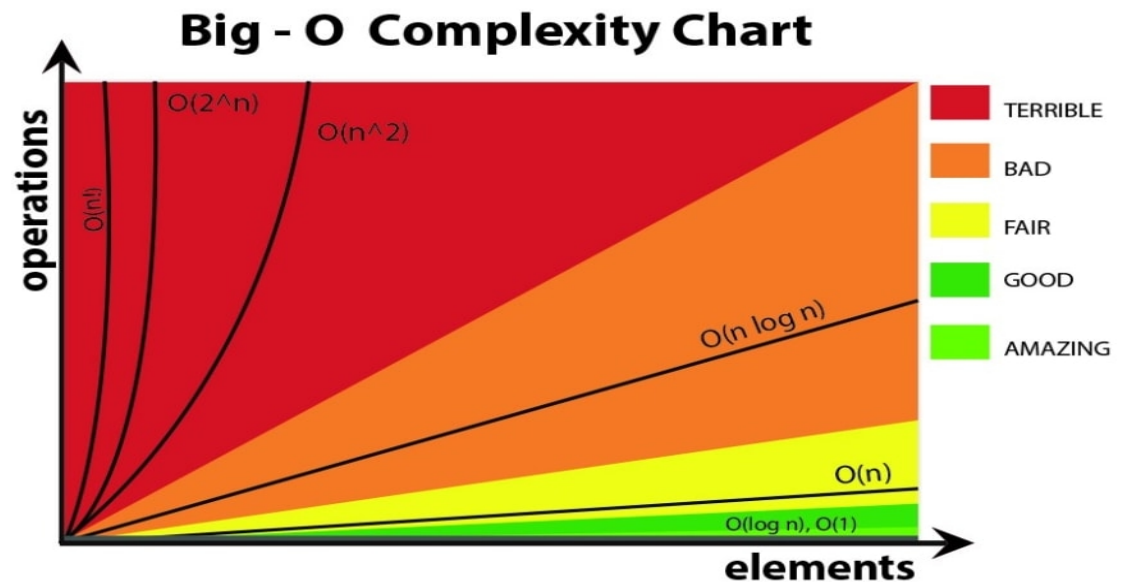
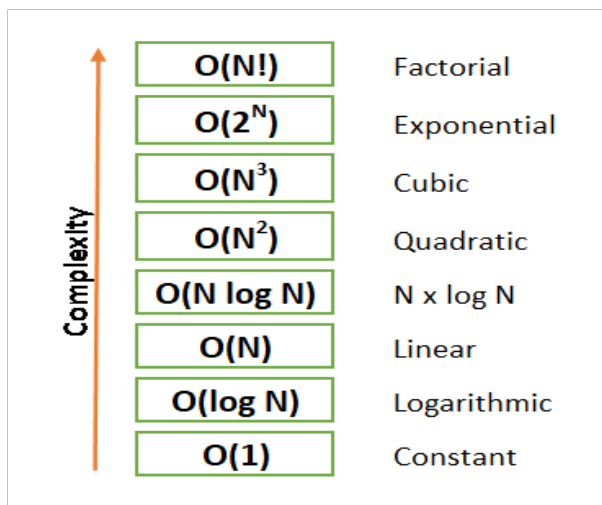
Complexidade Computacional

- **Complexidade Computacional mede o esforço de um algoritmo em computar um dado**
- **Problema**
 - **Quantos passos são necessários para encontrarmos um valor qualquer no vetor de tamanho N ?**



Notação BigO

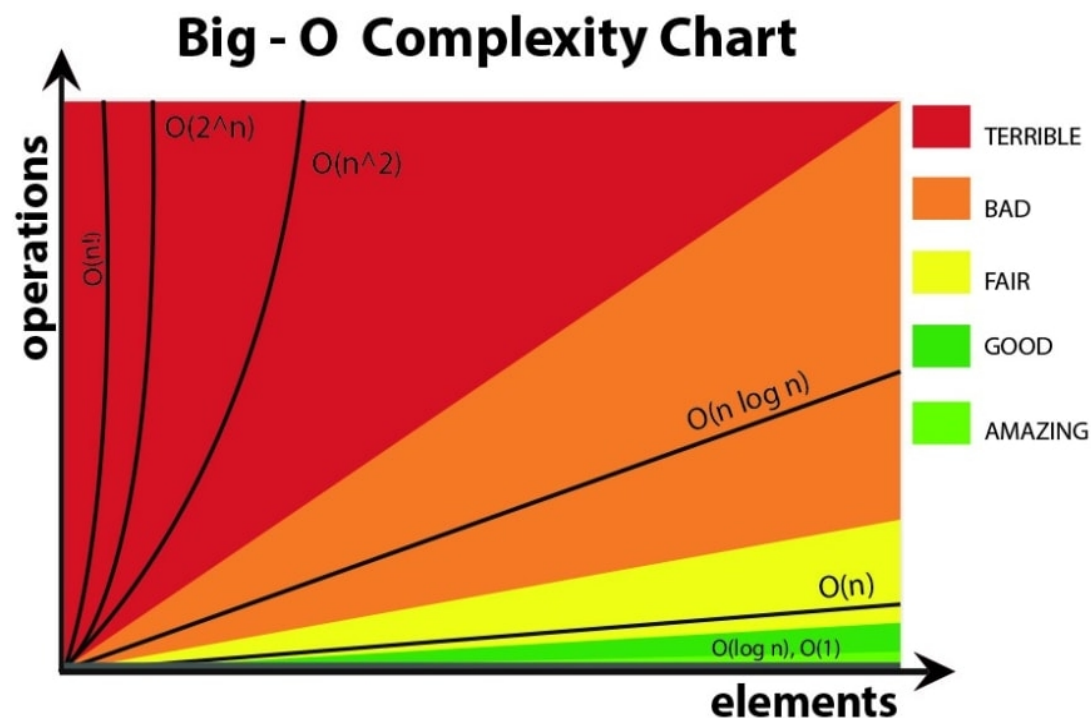
- **Análise Assintótica (Problemas Grandes)**
- **BigO $\rightarrow O()$**
 - **Formalização da complexidade de um algoritmo em razão da sua entrada (N).**
 - **Tempo ou Recurso em função de N**
 - **Considera-se o pior cenário para N**



Constante: $O(1)$

```
void printFirstElementOfArray(int arr[])
{
    printf("First element of array = %d",arr[0]);
}
```

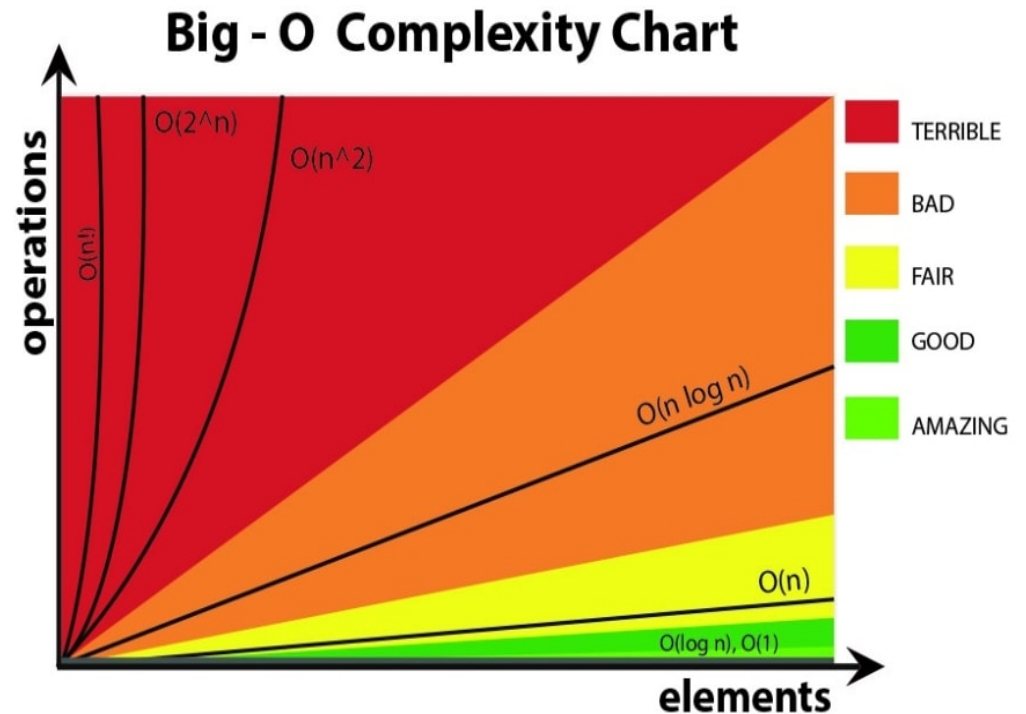
- $N = 1 \rightarrow 1 \text{ iter} \rightarrow O(1)$
- $N = 1000 \rightarrow 1 \text{ iter} \rightarrow O(1)$
- $N = 1000000 \rightarrow 1 \text{ iter} \rightarrow O(1)$



Linear: $O(n)$

```
void printAllElementOfArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d\n", arr[i]);
    }
}
```

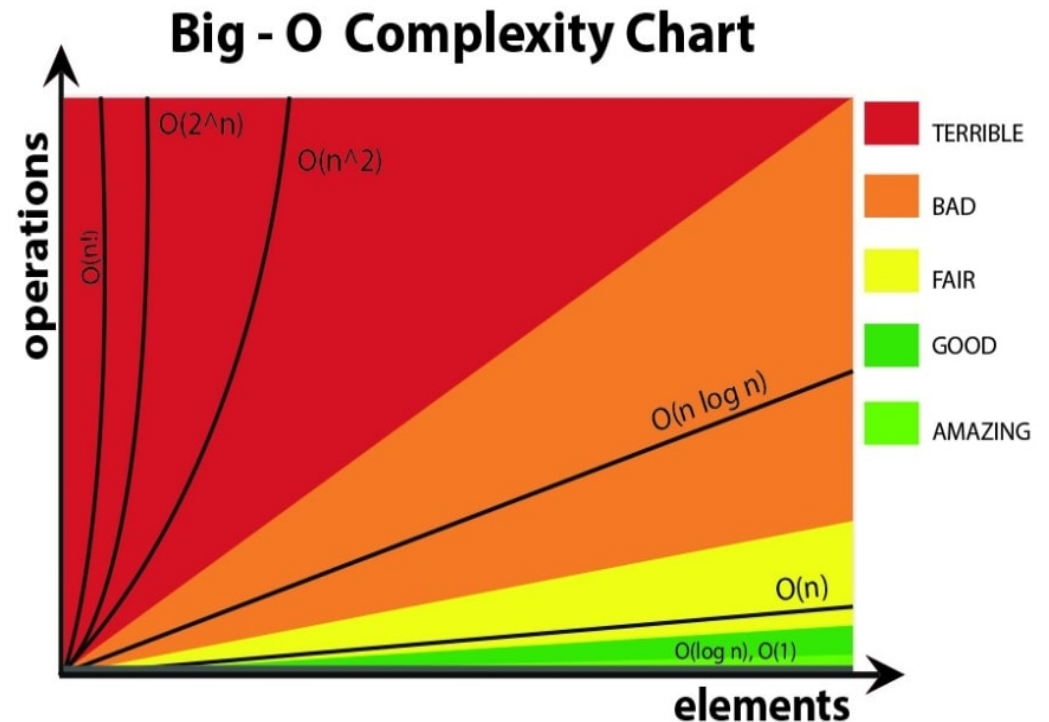
- $N = 1 \rightarrow 1 \text{ iter} \rightarrow O(N)$
- $N = 1000 \rightarrow 1000 \text{ iter} \rightarrow O(N)$
- $N = 10000 \rightarrow 10000 \text{ iter} \rightarrow O(N)$



Quadrática: $O(n^2)$

```
void printAllPossibleOrderedPairs(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            printf("%d = %d\n", arr[i], arr[j]);
        }
    }
}
```

- $N = 2 \rightarrow 4 \rightarrow O(n^2)$
- $N = 4 \rightarrow 16 \rightarrow O(n^2)$
- $N = 100 \rightarrow 10000 \rightarrow O(n^2)$
- **Cubica $\rightarrow O(n^3)$**
 - **3 laços aninhados**



Dicas ao computar BigO

- Elimine Constantes

$O(2n) \rightarrow O(n)$

```
void printAllItemsTwice(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d\n", arr[i]);
    }

    for (int i = 0; i < size; i++)
    {
        printf("%d\n", arr[i]);
    }
}
```

$O(1 + n/2 + 100) \rightarrow O(n)$

```
void printFirstItemThenFirstHalfThenSayHi100Times(int arr[], int size)
{
    printf("First element of array = %d\n", arr[0]);

    for (int i = 0; i < size/2; i++)
    {
        printf("%d\n", arr[i]);
    }

    for (int i = 0; i < 100; i++)
    {
        printf("Hi\n");
    }
}
```


Dicas ao computar BigO

- Ignore os termos/passos de menor relevância
 - $O(n + n^2) \rightarrow O(n^2)$
 - $O(n^3 + 50n^2 + 10000) \rightarrow O(n^3)$
 - $O((n + 30) * (n + 5)) \rightarrow O(n^2)$

```
for (int i = 0; i < size; i++)  
{  
    printf("%d\n", arr[i]);  
}  
  
for (int i = 0; i < size; i++)  
{  
    for (int j = 0; j < size; j++)  
    {  
        printf("%d\n", arr[i] + arr[j]);  
    }  
}
```

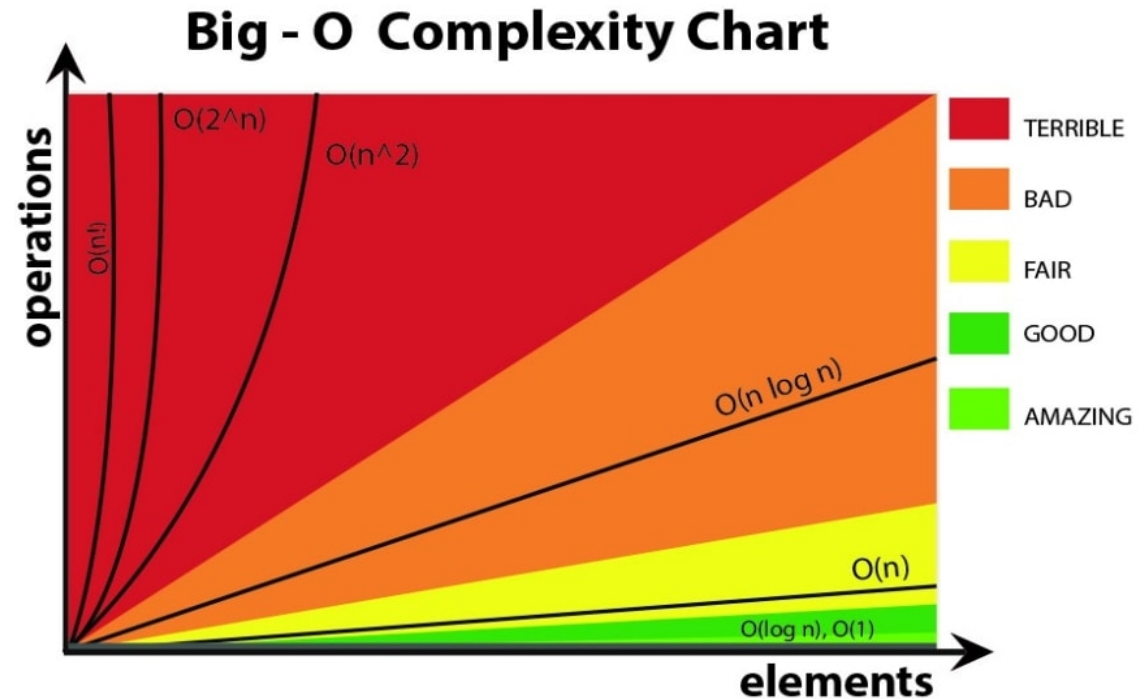
Codificando

- **DEEPNOTE LINK**

Logaritmica: $O(\log n)$

```
items = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
def binary_search(alist, item):  
    first = 0  
    last = len(alist)-1  
    found = False  
  
    while first <= last and not found:  
  
        midpoint = (first + last)//2  
  
        if alist[midpoint] == item:  
            found = True  
        else:  
            if item < alist[midpoint]:  
                last = midpoint-1  
            else:  
                first = midpoint+1  
  
    return found  
  
print(binary_search(items, 19))
```



- $N = 2 \rightarrow 1 \rightarrow O(\log n)$
- $N = 4 \rightarrow 2 \rightarrow O(\log n)$
- $N = 100 \rightarrow 10 \rightarrow O(\log n)$

$O(n \log n)$

```
for(int i = 0; i < n; i++)  
    for(int j = 1; j < n; j = j * 2)  
        print(i, j)
```

- $O(n) * O(\log n) == O(n \log n)$

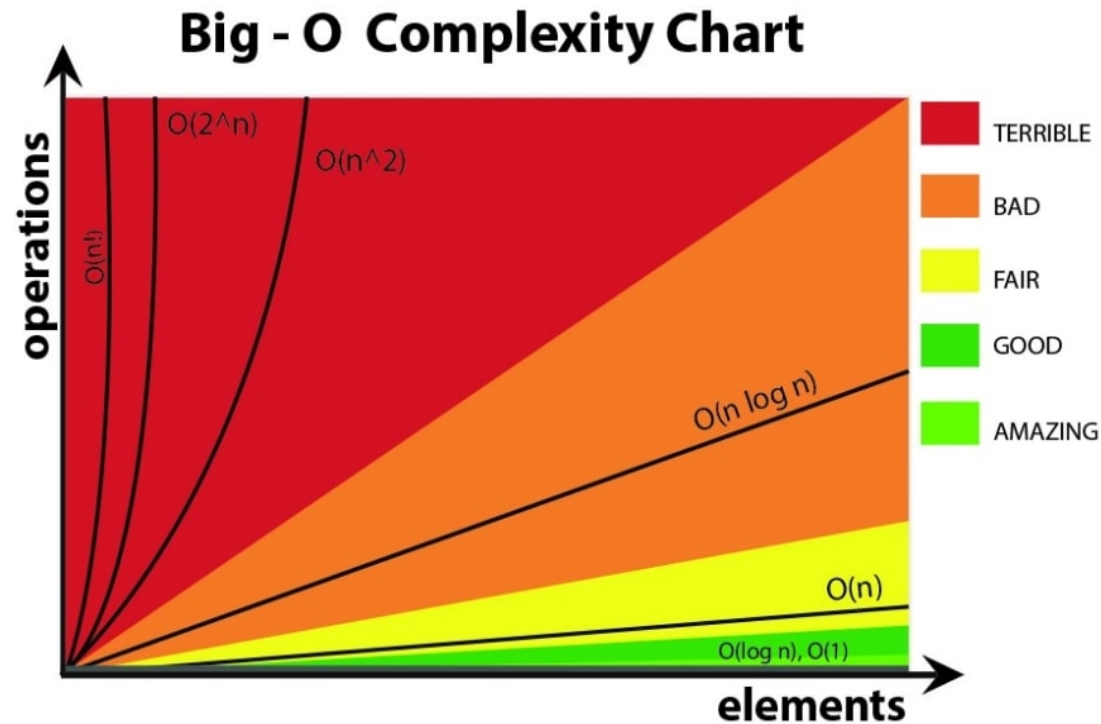
N=2 -> 2

N=4 -> 8

N=6 -> 18

N=8 -> 24

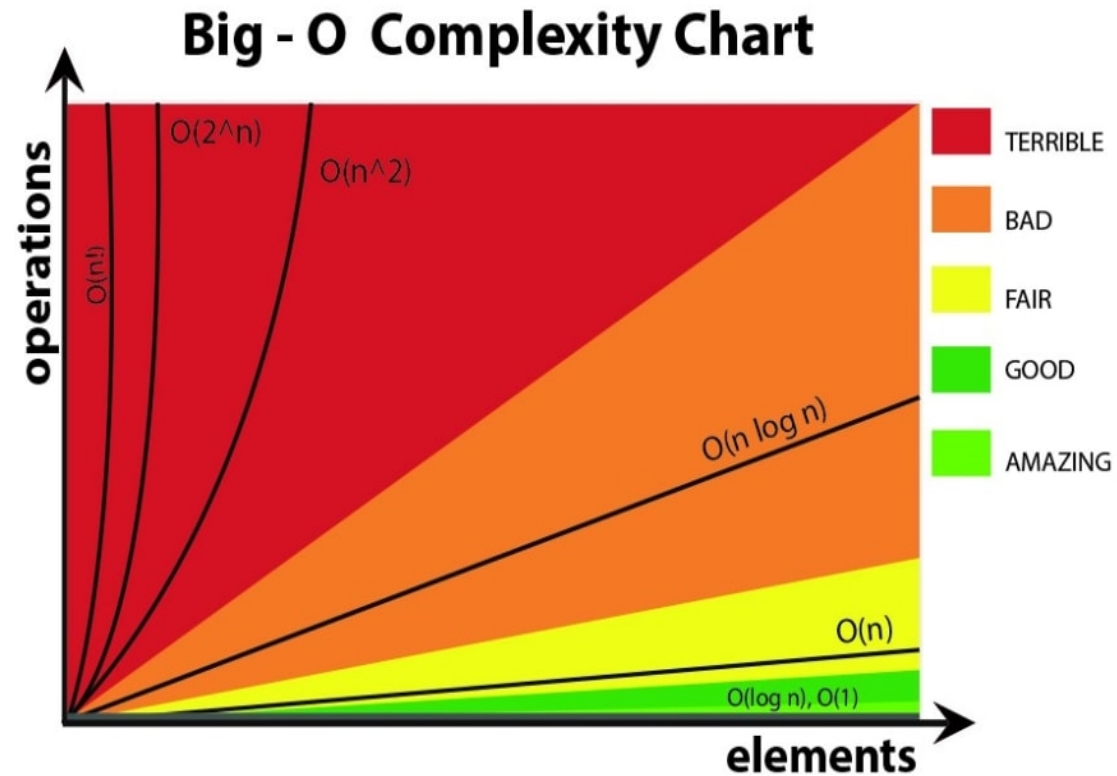
N=10 -> 40



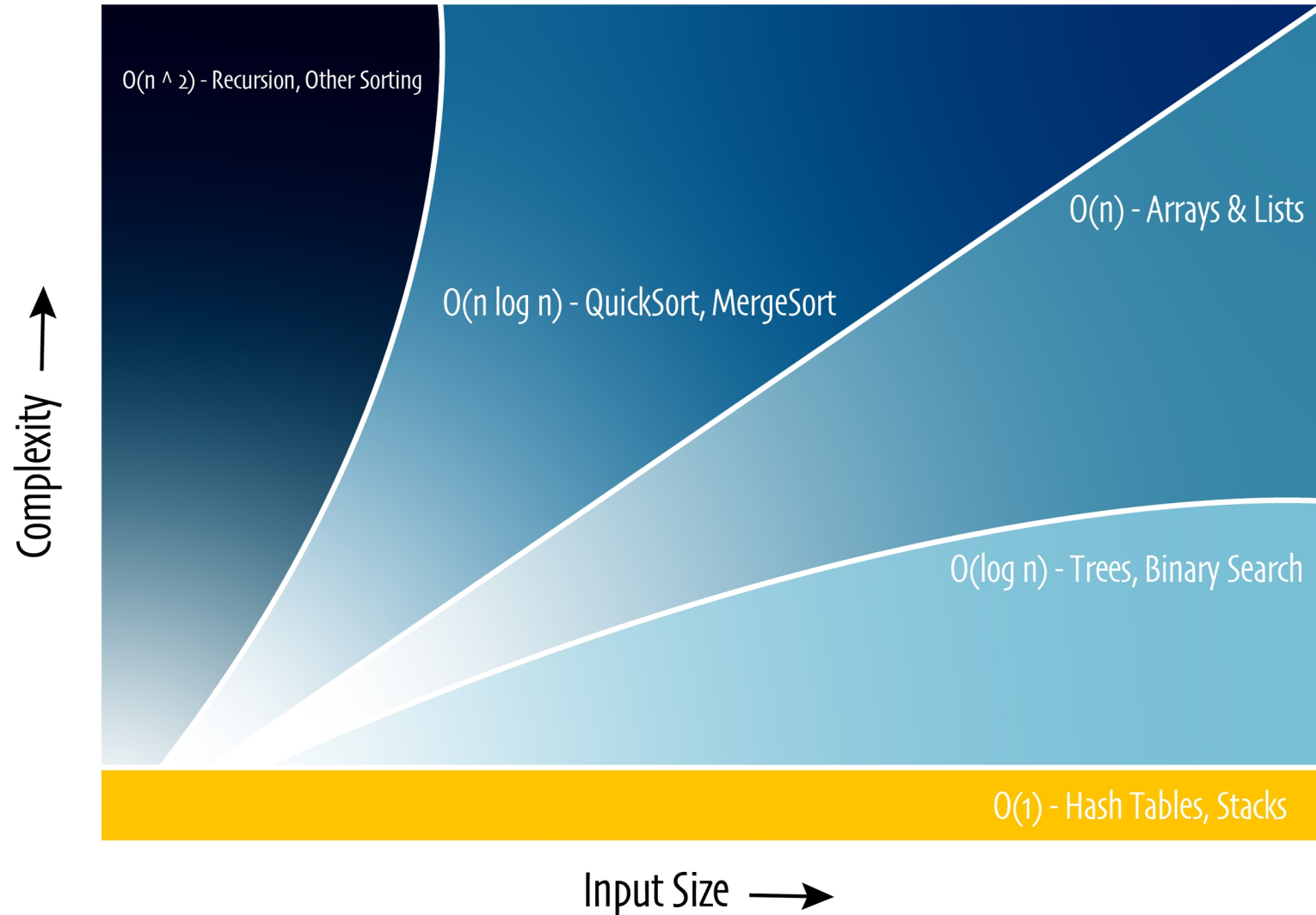
Exponencial: $O(2^n)$

```
int fibonacci(int num)
{
    if (num <= 1) return num;
    return fibonacci(num - 2) + fibonacci(num - 1);
}
```

- $N = 2 \rightarrow 4 \rightarrow O(2^n)$
- $N = 4 \rightarrow 256 \rightarrow O(2^n)$
- $N = 10 \rightarrow 1024 \rightarrow O(2^n)$



Big O - Estruturas de Dados

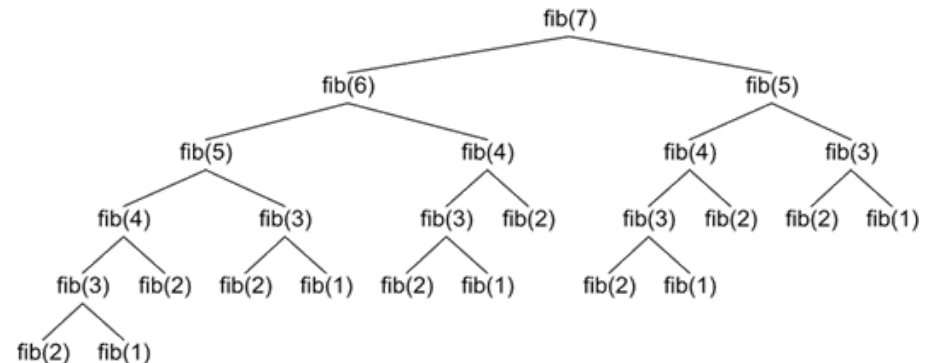


Programação Dinâmica

- Richard Bellman , 1950
- ‘Programação’ tem sentido de ‘Planejamento’
- Problemas recursivos com sobreposição de subproblemas
- Utiliza uma estrutura computacional (vetor/matriz) para armazenar resultados dos sub-problemas
- Ex: Fibonacci recursivo é ineficiente (Exponencial)

$$F(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ F(n-1) + F(n-2) & \text{caso contrário} \end{cases}$$

```
int fibonacci(int num)
{
    if (num <= 1) return num;
    return fibonacci(num - 2) + fibonacci(num - 1);
}
```



- $O(2^n)$

Programação Dinâmica

- **Solução: Armazenar subproblemas já computados**
- **$O(n)$**

```
def FibonacciDP(n, computed={0:0, 1:1}):  
    if n not in computed:  
        computed[n] = FibonacciDP(n-1, computed) + FibonacciDP(n-2, computed)  
    return computed[n]
```

- **Codificar no deepnote!**

Programação Dinâmica - Trabalho

- **Trabalho: Maior Subsequência Comum (Longest Common Subsequence)**
 - **Encontrar o comprimento da maior subsequência entre duas sequências. Subsequência é uma sequência que aparece na mesma ordem, porém não necessariamente contígua.**
 - **Input “ABCDGH” e “AEDFHR” gera “ADH” (tam = 3)**
 - **Input “AGGTAB” e “GXTXAYB” gera “GTAB” (tam = 4)**
 - **Solução Recursiva:**

```
def lcs(X, Y, m, n):  
  
    if m == 0 or n == 0:  
        return 0;  
    elif X[m-1] == Y[n-1]:  
        return 1 + lcs(X, Y, m-1, n-1);  
    else:  
        return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));  
  
# Driver program to test the above function  
X = "AGGTAB"  
Y = "GXTXAYB"  
print "Length of LCS is ", lcs(X, Y, len(X), len(Y))
```

Seminário Final - Individual

- **Na próxima aula apresente um seminário de 10 a 20 minutos sobre:**
 - **Como a estrutura de dados está presente na sua área de atuação ?**
 - **Apresente o que você está levando da disciplina para o dia-a-dia**
 - **Você passou a enxergar essas estruturas, mesmo que de forma indireta?**
 - **Na sua área de pesquisa, você enxerga o uso delas? Ainda que de forma indireta?**
 - **.....**